U.S. PATENT APPLICATION

FOR

SYSTEM, METHOD AND COMPUTER PROGRAM PRODUCT FOR RECOGNIZING A STATE OF A REMOTE APPLICATION

INVENTORS: DMITRIY YEPISHIN
ASAD ZAIDI
VLADIMIR SOSKOV
QUINTON ZONDERVAN
SHAILESH GARG
HARI KRISHNA ANNAMARJU
RODOLFO FERNANDEZ
JONATHAN ROCHEZ

ASSIGNEE: CLICKMARKS, INC.

System, Method and Computer Program Product for Recognizing a State of a Remote Application

Dmitriy Yepishin
Asad Zaidi
Vladimir Soskov
Quinton Zondervan
Shailesh Garg
Hari Krishna Annamarju
Rodolfo Fernandez
Jonathan Rochez

FIELD OF THE INVENTION

The present invention relates to computer-related transactions, and more particularly to automating computer-related transactions.

BACKGROUND OF THE INVENTION

The Internet is composed of content distributed in the World Wide Web and various intranets. While a large fraction of the content is static, the truly interesting content is the one that a user can interact with dynamically. This content is of various types including, but not limited to (i) the content stored in various databases, (ii) e-commerce web-pages, (iii) directories, (iv) intranet pages, (v) data warehouses, etc.

The interaction with this dynamic content is accomplished through (i) queries/submissions to databases, (ii) buying/selling/interacting through e-commerce, (iii) running queries and lookups in directories, (iv) accessing and interacting with content

resident on intranet pages (including on individual computers), and/or (v) accessing, interacting with, adding, subtracting or modifying content resident in data warehouses.

The access to or interaction with this dynamic content is done in a variety of ways. For example, such interaction may be accomplished through direct access to the databases by running specific commands or through form submissions on the Internet that run specific queries or perform specific actions. This interaction requires the submission of necessary parameters or information to complete a query or interaction (addition, modification, subtraction) with the dynamic content. This information may need to be submitted in multiple steps. Once the submission of information is finished, the results of the interaction/query/e-commerce are sent back to the user.

Each time a user wishes to interact in the foregoing manner, the user is required to carry out each and every one of the steps associated with the submission of necessary parameters or information. If a same type of transaction is to be carried out in a repeated manner, this may be very time consuming and problematic.

Accordingly, accessing web content is more complicated than simply making individual HTTP requests. The prior art has yet to enable fetching of the same content as the user and rendering it the same way the user saw it. To do this, the appropriate content must first be fetched across the network. It must then be rendered correctly.

When fetching the content, the user may first be required to log in or run a search for a certain term. More generally, the content of interest could be generated by an arbitrary web transaction. Logging in and running a search are all examples of web transactions. Thus, fetching content requires support for various authentication and network protocols, management of client-side state information as well as support for the appropriate cipher strength.

It should be noted that fetching any interactive web content requires the ability to be able to execute web transactions. In the case of non-interactive content (e.g. the top headlines

from a news site), no transaction is required to retrieve the content. One simply has to request the page from the remote server. However, if any interaction is required to access that content (e.g. weather report for a particular zip code), the transaction must be executed before the content can be retrieved.

5

Web transactions vary in their complexity. They may be as simple as entering a zip code to receive a customized weather report. On the other hand, they may be complex enough to involve logging in to a secure stock trading site, browsing to a particular page on the site, submitting a query and then browsing to a specific section in the report to obtain the

10   current credit rating of a company.

Rendering the content is also a challenging problem. To start with, there is no well-defined standard for how web pages *should* display. Identical HTML is often rendered differently by competing browsers. Secondly, even apart from standards disputes and

15   ambiguities, one can never be entirely sure that one has rendered a page as intended by the page designer. The designer may not have intended it for display on certain resolutions. Furthermore, not all browsers and clients are equally capable.

The aforementioned problems are compounded when attempting to access and navigate

20   websites using wireless devices. Most websites are not wireless enabled. Nor can most wireless devices access and retrieve content from the websites directly.

There is therefore a need to allow wireless enablement of any website to allow output of desired content on a wireless device in a format amenable to display on the wireless

25   device. There is also a need for a method to render active content into a format amenable to display on a device that does not support active content.

There is also a need for a way to retrieves such content as a user's electronic "agent." More particularly, what is needed is an electronic user agent that attempts to make

30   reasonable compromises so that it can fetch and render popular web content correctly and with acceptable performance.

CLIC1P019

What is a further need a way to replay patterns so that active content can be output on a wireless device without requiring that the entire web page be displayed. There is also a need for a way to complete transactions on behalf of a user using a pattern, regardless of

5   which state the website is in at the time the request is initiated. In other words, there should be a mechanism to recognize when a user strays from the pattern and reinitiate the pattern at the new state.

Summary of the Invention

A system, method, and computer program product are provided for recognizing a state based on a current output, according to one embodiment of the present invention. Current output is retrieved from a remote application utilizing a network, the current output corresponding to a state of the remote application. An output structure description of the remote state is generated. The output structure description is preferably of a window and frame structure of the output. The output structure definition is compared to a pre-defined state definition of a recorded state (called a state description) for determining whether the state of the remote application corresponds to the recorded state. Note that the recorded state definition can be created during recording of the pattern.

As an option, predefined information (e.g. a StateData Object) about the remote state can be returned if the remote state corresponds to the recorded state. The predefined information can then be used to extract data from the remote application.

In one aspect of the present invention, the output structure definition includes a number of windows presented in the remote state, a number of frames in each window, a nesting structure of the windows, and/or a nesting structure of the frames. Preferably, the recorded state is part of a pattern pre-recorded by a user. In yet another aspect of the present invention, content of the remote output is recognized based on recorded content properties and used to help recognize the state. Preferably, the recognized content is utilized for recognizing the state based at least in part on a weighted average of content recognition and content importance. Note that the content can be one or more documents, one or more content items in a document(s), etc.

A method is provided for recognizing a state of a remote application utilizing individual document recognition, according to another embodiment of the present invention. First, a plurality of documents are received. An attempt to recognize each of the documents as being associated with a predetermined state is made. The state of the remote application is identified based on the recognition of the documents.

A Content Recognition Module (CRM) can be used to recognize states. Basically the state consists of multiple documents and/or content items, each one of which is first recognized by the CRM. If the state consists of documents A, B and C, then if the CRM recognizes all of A, B and C from the documents given it by the SRM, the state is recognized. Otherwise, if the CRM recognizes a different set of documents, then the state is recognized as the state containing that set of documents. No state is recognized if no state corresponds to that set of documents, the pattern can be placed in an error state.

More information about operation and functionality of the Content Recognition Module is found in US Patent Application entitled SYSTEM, METHOD AND COMPUTER PROGRAM PRODUCT FOR CREATING A DESCRIPTION FOR A DOCUMENT OF A REMOTE NETWORK DATA SOURCE FOR LATER IDENTIFICATION OF THE DOCUMENT AND IDENTIFYING THE DOCUMENT UTILIZING A DESCRIPTION, filed concurrently herewith and assigned to common assignee Clickmarks, Inc., and which is herein incorporated by reference.

Preferably, if more than one pattern state matches the remote state, a comparison of output structures of the remote state and pattern states is performed, as set forth above.

In one aspect of the present invention, at least a portion of the state definitions are given a weighting based on a probability that they will occur when in another state. Preferably, the weighting is based on a state resulting from execution of a prior action. For example, if

during last replay (or record) of a pattern, action x executed in state A resulted in state B, then during current replay when x is executed in A, B will be the most probable resulting state, more so than any other state in the pattern. The weight is used to optimize the state recognition algorithm by comparing the state to be recognized first with the state most likely to occur after executing the action in the previous state. If that state does not match the state to be recognized, then the next most likely state is compared, and so on. Content item weights are defined on the document as it is part of the state, but they are not defined on the state itself.

5

10

BRIEF DESCRIPTION OF THE DRAWINGS

Figure **1** illustrates a typical hardware configuration of a workstation in accordance with

5   a preferred embodiment;

Figure **2** illustrates a system for navigating a network, including conducting transactions, in accordance with one embodiment of the present invention;

10   Figure **3** is a flowchart of a process for User Agent operation according to an embodiment of the present invention;

Figure **4** illustrates a flow of User Agent operation at design-time;

15   Figure **5** shows the flow of User Agent operation at runtime;

Figure **6** is a flow chart that illustrates a method for recording a pattern using the User Agent according to one embodiment of the present invention;

20   Figure **7** illustrates a recording interface according to one embodiment of the present invention;

Figure **8** illustrates a mode selection screen of the interface of Figure **7**;

25   Figure **9** shows a field selection screen of the interface of Figure **7**;

Figure **10** depicts a filter screen of the interface of Figure **7**;

Figure **11** illustrates a results screen of the interface of Figure **7**;

CLIC1P019

Figure **12** illustrates an assign variable screen of the interface of Figure **7**;

Figure **13** shows the grid screen of the interface of Figure **7**;

5

Figure **14** shows the grid screen of Figure **13** with additional objects added;

Figure **15** illustrates a display specification screen of the interface of Figure **7** from which the user can specify the manner in which variables are displayed on particular

10 devices;

Figure **16** gives a general overview of how the PRE interfaces with other major components of the system according to one embodiment of the present invention;

15 Figure **17** is a flowchart of a process for replaying a pattern for transitioning from one state to another state, according to an embodiment of the present invention;

Figure **18** depicts a process for replaying a pattern for transitioning from one state to another state in a remote application, according to another embodiment of the present

20 invention;

Figure **19** is a flowchart of a method for identifying a dynamic element and replaying a dynamic event according to an embodiment of the present invention;

25 Figure **20** illustrates a Connector class hierarchy according to one embodiment of the present invention;

Figure **21** depicts a State class hierarchy according to one embodiment of the present invention;

CLIC1P019

Figure **22** is a diagram illustrating how different components of the present invention can interact to support Scripting;

5   Figure **23** depicts a process for recognizing a state based on a current output, according to one embodiment of the present invention;

Figure **24** is a flow diagram of a process for recognizing a remote state utilizing individual document recognition, according to another embodiment of the present

10   invention;

Figure **25** is a flow diagram that illustrates the interaction between the XML Connector Module (XCM) and the Pattern Replay Engine (PRE);

15   Figure **26** is a flow diagram of the operation of the XML Connector;

Figure **27** is a flow diagram of SRM and IDT interaction; and

Figure **28** depicts a program flow of the Connector.

20

CLIC1P019

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Glossary

5

action   An event which can be executed by the user or by script to change the state of the remote application (thus changing the state of the local application). For example, clicking a link.

content   Any content. Also includes a sub-tree of a full DOM tree.

client   A process which makes requests to and, presumably, gets web pages from the User Agent.

connector   An internal module of the platform, which can execute a state in a pattern recorded by the IDT.

CRM   Content Retrieval Module, a component of the platform.

custom (local) An application created utilizing the platform
application

DOM   Document Object Model, a W3C standard for describing XML documents in an object-oriented fashion.

CLIC1P019

documents in an object-oriented fashion.

DTD   Document Type Definition, a document used to define an XML markup language. It contains the rules by which an XML Document of the corresponding markup language is constructed/validated.

element   An XML element. Everything from `<tag>` to `</tag>`

HTML connector   A component of the platform. This component calls the User Agent directly. This is also named the XML connector.

IDT   Interactive Design Tool, the design tool of the platform. Allows a user to create patterns and custom application layouts.

pattern   A collection of states grouped together by a platform user to be used as one functional element in the custom application.

platform   The platform of the system.

PRE   Pattern Replay Engine, a component of the platform that replays patterns recorded by the IDT.

remote application	An application, which exists on some remote site and has some functionality of interest that the user of the platform desires to extract from it. (For, example Yahoo Mail is a remote web application).
remote state (remote output)	Corresponds to a stable output from a remote application at some point in time. (In case of Yahoo Mail, the login page is one state and the page, which displays the Inbox, is another state.)
SessionManager	A class responsible for keeping track of active sessions and associated data such as Patterns and Variables.
SRM	State Recognition Module, a component of the platform. Provides state recognition and content retrieval functionality.
state	State of the remote application as defined by the user of the platform (the designer). (This will usually correspond to a given remote state; however, since not all parameters of the remote system are known, the local state cannot have all of the information that is available to the remote state.) States are contained within a *Pattern* (see definition).
step	A Step is part of a larger sequence of mappings to States. The list of Steps serves as a history of the States traversed by the PRE. The

reason for using Steps is the user may go "Back" to a previous State in the Pattern and initiate an Action from that particular point in the traversal history.

UA
The User Agent, a component of the platform. Used to fetch the output from a remote application and execute any user actions on that output.

variable
Represents a value which the designer's scripting code can manipulate. Variables can be one of the following three scope types: User, Session, or Application. And the Variable's underlying value can be one of three value types: (1) Primitive (not strongly-typed) (e.g. int, string, boolean), (2) Record (which is like a struct), and (3) Table (a list of records).

web content
See *content*.

XHTML
Extensible HyperText Markup Language, an XML-compliant version of HTML. XHTML is viewable on major browsers.

XML
Extensible Markup Language, a syntax for creating SGML-compliant markup documents. The rules by which a document is constructed/validated can be specified via a DTD or XML Schema. XHTML is an example of an XML compliant markup language.

XML documents may also be created which do not correspond to an explicitly defined schema. Such documents are said to be well-formed if they conform to the syntactical rules of XML, but their overall structure can be arbitrary.

XML connector   An internal module of the platform, which can execute an Extensible Markup Language (XML) state in a pattern recorded by the IDT. Also called the HTML connector.

Illustrative System Architecture

5   Figure 1 illustrates a typical hardware configuration of a workstation in accordance with a preferred embodiment having a central processing unit 110, such as a microprocessor, and a number of other units interconnected via a system bus 112.

The workstation shown in Figure 1 includes a Random Access Memory (RAM) 114,

10   Read Only Memory (ROM) 116, an I/O adapter 118 for connecting peripheral devices such as disk storage units 120 to the bus 112, a user interface adapter 122 for connecting a keyboard 124, a mouse 126, a speaker 328, a microphone 132, and/or other user interface devices such as a touch screen (not shown) to the bus 112, communication adapter 134 for connecting the workstation to a communication network

15   135 (e.g., a data processing network) and a display adapter 136 for connecting the bus 112 to a display device 138.

The workstation typically has resident thereon an operating system such as the Microsoft Windows NT or Windows Operating System (OS), the IBM OS/2 operating

CLIC1P019

system, the MAC OS, or UNIX operating system. Those skilled in the art may appreciate that the present invention may also be implemented on platforms and operating systems other than those mentioned.

5   Figure **2** illustrates a system **200** for navigating a network **202**, including conducting transactions, in accordance with one embodiment of the present invention.

A Request Handler (RH) **204** communicates with a user device **205**. The RH manages requests from the user device, routing them to the appropriate system component.

10   When a user requests a transaction, the request is sent to a Pattern Replay Engine (PRE) **206**, which replays a pattern for conducting a transaction on behalf of a user. More information about the PRE is set forth below.

The State Recognition Module (SRM) **208** determines which state a website is in based

15   on its current output, such as a structure of the current output. The SRM may communicate with a Content Recognition Module **210**, which recognizes individual documents that can comprise a state. A Connector **212** is in communication with the SRM. The Connector executes a state in the pattern. The SRM, Content Recognition Module, and connector are described in detail below. More information about operation

20   and functionality of the Content Retrieval Module is found in US Patent Application entitled SYSTEM, METHOD AND COMPUTER PROGRAM PRODUCT FOR RETRIEVING ARBITRARY CONTENT FROM A REMOTE DATA SITE FOR OUTPUT TO A DEVICE IN A FORMAT SUITABLE FOR THAT DEVICE, filed concurrently herewith and assigned to common assignee Clickmarks, Inc., and which is

25   herein incorporated by reference.

The User Agent **214** is used by other components of the system to provide the actual interaction with a remote website. For example, when replaying a pattern, the SRM communicates with the User Agent via the Connector to provide instructions to the User

Agent. The other system components have intelligence built into them that instructs them how to utilize the User Agent. For example, when a user clicks on a button on a page, other components instruct the User Agent to navigate to the desired web page and perform some action , such as filling in a form. The User Agent retrieves the resulting

5   page and returns it to the other components.

By default, the User Agent is not running. A listener (not shown) listens for requests. When the listener receives a request, it creates a new User Agent process on the server and returns an identifier that identifies the User Agent process. Subsequently, client

10   processes use the identifier, go to the specific User Agent and instruct it to perform some action. The User Agent performs the action according to the instructions and returns the results of the action.

A Transcoding Page Rending Engine (TRE) **216** renders content for display on the user

15   device. Preferably, the TRE is able to render content on any display environment. More information about operation and functionality of the TRE is found in US Patent Application entitled SYSTEM, METHOD AND COMPUTER PROGRAM PRODUCT FOR PAGE RENDERING UTILIIZING TRANSCODING, filed concurrently herewith and assigned to common assignee Clickmarks, Inc., and which is herein incorporated by

20   reference.

In the present invention, a transaction preferably refers to communicating (i) information and/or actions required to conduct the transaction, and/or (ii) information and/or actions sent back or desired by the user, respectively.

25

For example, a transaction, in one embodiment, may refer to: information submitted by the user, actions taken by the user, actions taken by a system enabling the access of the user to the data, actions taken by the data to retrieve/modify content, results sent back to the user, and/or any combination or portion of the foregoing entities.

User Agent

5   The User Agent is able to fetch web pages of interest and render them correctly. The operation of the User Agent can be likened to remote controlling navigation of a browser on a remote server. For example, the User Agent acts as an intermediary on a remote server that can be commanded to go online and retrieve content, and then send it for rendering for output on a wireless device. The system component that renders the

10   content for output on the wireless device is the Transcoding Page Rendering Engine (TRE).

The User Agent can be used both via a network interface and Interprocessor Communication (IPC). Furthermore, it has performance and scalability befitting a

15   server application.

Figure 3 is a flowchart of a process 300 for User Agent operation. In operation 302, a network site is accessed utilizing a network. This can be done by navigating to a particular web site using its URL, for example. In operation 304, a user interaction is

20   simulated for navigating the network site based on client specifications (i.e., directions). An example of such navigation is logging in to a web site. The client is a process that makes requests to and receives output from the Use r Agent. A web page is retrieved from the site in operation 306. Such content can be part or all of a web page, for example, such as the resulting page that is accessible upon logging in to a web site. In

25   operation 308, identifiers are assigned to elements of the web page. For example, the third element on the page could be assigned the ID "3". The web page is rendered for further processing utilizing the identifiers in operation 310. The rendering can include converting the resulting page into XHTML for processing by another system component and/or output on a user device. Note that XHTML is used herein by way of

CLIC1P019

example; any other XML compliant format may be used; that is, if the page loaded by the User Agent is an XML page, as opposed to an HTML page, it is not converted to XHTML, but rather is passed on to the client in its original XML format. In operation **312**, dynamic content of the web page such as scripting is rendered utilizing the

5   identifiers. The rendered web page is returned to the client in operation **314**. As noted above, the rendered content can be output to a user device, such as a wireless device. The wireless device can include a wireless telephone, Personal Digital Assistant (PDA), handheld computer such as a handheld PC, a pager, a device connected to a wireless modem, or any type of device capable of receiving information where at least one of the

10   communication links is wireless.

In one aspect of the present invention, navigation from the web page is performed. For example, navigating from the web page can include of clicking links on a web page, submitting forms, and/or firing scripting events (clicking a link being a specific kind of

15   event).

In another aspect of the present invention, the client is informed of changed frames. The User Agent is able to inform the client which frame(s) have changed since the last event was executed. Navigating to a specified URL would be considered an event in

20   this context. Typically, this would be used by the client to determine which frames have changed since the last event was executed and it would then request only the modified frames. In a further aspect, specified frames are sent to the client. The User Agent is ideally able to send any frames which are requested to the client. Often, a web page will comprise a single frame. However, if the page comprises multiple frames, the

25   User Agent is able to send a selected single frame (or a specified subset of the frames) to the client.

In a further aspect, a network interface is provided. A runtime engine can use a network interface to make requests to the User Agent. It uses the network interface to navigate

the User Agent and to request the web pages from it. As an option, a Component Object Model (COM) interface can be provided. In a preferred environment, the User Agent is embedded as a COM object. The embedding application can then interact with the User Agent's browser component directly. It may then also request the XHTML

5   representations of the current web page. Again, XHTML is used herein by way of example, and any other XML compliant format can be used.

Module dependencies

10   The User Agent module is not required to use other modules from the platform. It is used by the HTML connector and the IDT.

Technologies used

15   The User Agent is preferably a Windows application. It can use COM and standard Microsoft browser libraries (i.e. MSHTML.DLL). In addition, it can use standard libraries for common functionality (e.g. HTTP communication).

Functionality

20

The User Agent is able to fetch the web pages and render them "correctly". While there is no universally agreed upon definition of "correct", the User Agent attempts to conform to de facto standards. The User Agent acts, to the degree possible, like a web browser running on a desktop machine and operated by a single user. Broadly

25   speaking, a preferred embodiment of the User Agent:

- satisfies all protocol requirements (e.g. HTTP, FTP, SSL and TLS),
- supports all technologies of interest (e.g. JavaScript, VBScript),
- renders pages in a generally accepted manner

- shares no state information with any other User Agent instance.

Remote web sites should not be able to determine whether they are interacting with a human user on a dedicated machine or the User Agent. For example, two different User
5   Agents running on the same server should be able to simultaneously log-in to Hotmail using two different user names (and get two different pages).

Constraints

10 The User Agent as described herein is designed for use at design-time and runtime. The two usage scenarios impose different constraints.

Design-time usage

15 At design-time, the User Agent is run locally on the designer's machine (embedded directly in the Development Tool (DT) used by the designer). It exposes the browser interface directly to the DT. See also Figure 4 and the related discussion below for more information.

20 *Runtime usage*

At runtime, the User Agent is run remotely on a dedicated server. A network interface is exposed so that remote clients can interact with the UA. HTTP can be used to provide this interface. As it is a very widely used protocol, it allows for a very flexible
25 system design (as many diverse clients can speak HTTP) and there are a plethora of standard tools (e.g. performance auditing, load balancing, security tracking) available for it. See also Figure 5 and the related discussion below for more information.

Two interface methods: HTTP and COM

The User Agent can support a COM interface so that it can be embedded into the DT at runtime. It can also support an HTTP interface so that it can be used by remote clients

5   at runtime.

Performance and scalability

The User Agent can be used primarily as a server-side application which services client

10   requests. Each User Agent server is able to support several hundred simultaneous users with acceptable response time. In addition, it is possible to increase the number of users which can be reasonably supported by adding more hardware to the server set-up.

Functionality

15

Abstraction

The User Agent is able to navigate to a desired web page (rendering any dynamic content) and then send an XHTML representation of the rendered web page (which may

20   comprise several frames) to the client. It also allows for the client to be able to direct it to a specified URL, click links, submit forms and fire scripting events on the page. Preferably, it is able to: Note that XHTML is used herein by way of example, and any XML compliant format can be used.

25   1. *Navigate to a specified URL and retrieve the associated page over the network.*
           This is the process of actually making network requests and receiving responses.
           It involves handling all required network protocols, including protocol
           requirements such as maintaining client-side state information (e.g. HTTP
           cookies) and having appropriate security certificates.

2. *Render those pages, tag them and convert them to XHTML.* Once the page has been fetched, any dynamic elements (e.g. scripting) are rendered in XHTML for output on the device. While JavaScript is the scripting language of choice, any

5 common scripting language (such as VBScript) could be used. The User Agent also generates and assigns unique IDs to each element on the page. It can use the integer indices of the elements on the page for the IDs (starting from one). For example, the third element on the page would be assigned the ID "3". It then converts the resulting page into XHTML if it is an HTML page. To render

10 a page, the UA executes any scripting which is to be executed upon the page loading (prior to any user actions). Any scripting which is run as a result of a user action (for example, onMouseOver) is not executed at this point.

Preferably, no scripting is removed or modified from the page. Thus, if a naïve

15 client loads the page into a script-capable browser, it will then execute any automatic events twice. In addition, if an event is fired, whatever script would have been originally called is executed. If the client wishes to modify these events (so that, for example, onClick calls a script of the client's choice instead of the original script), the client modifies the script.

20

3. *Navigate from the current web page as specified by other Platform components.* This includes clicking links on a web page, submitting forms and firing scripting events. This comprises two parts: filling in forms and firing events (clicking a link being a specific kind of event). Given a specified event (complete with all

25 the parameters needed to execute that event, including the element on which the event is to be fired), the User Agent executes the event on the web page. Typical events include onClick and onSubmit.

4. *Inform the client of changed frames* – The User Agent, upon request, informs the client which frame(s) have changed since the last event was executed. Navigating to a specified URL can be considered an event in this context. Typically, this would be used by the client to determine which frames have changed since the last event was executed and it would then fetch only the modified frames.

5. *Send specified frames to client* – The User Agent can be able to send any frames which are requested to the client. Often, a web page will comprise a single frame. However, if the page comprises multiple frames, the User Agent is able to send only a single frame (or a specified subset of the frames) to the client.

6. *Provide a network interface.* The runtime engine uses a network interface to make requests to the User Agent. It uses the network interface to navigate the User Agent and to request the web pages from it.

7. *Provide a COM interface.* The design-time environment can embed the User Agent as a COM object. It is then able to interact with the browser directly. It may then also request the XHTML representations of the current web page.

8. *Support scaling easily to multiple server configurations.* It is possible to add servers to a UA server farm easily, quickly and without any of interruption of service.

The following sections detail the HTTP interface and the COM interface.

HTTP interface

When running in server mode, the User Agent exposes an HTTP interface. Clients can create new sessions, control existing sessions, retrieve the content of various web pages and terminate sessions via the HTTP interface.

Session creation

Clients make HTTP POST requests to a well-known URL (e.g. http://user_agent:80) to spawn new UA instances. A new UA instance is spawned and assigned a port number. The body of the response contains the server name and this newly assigned port number. The concatenation of the protocol used, the server name and the port number are called the *assigned URL*. Clients can then make future requests to the assigned URL.

Clients may specify a timeout period (in seconds) in the body of the response. If the UA instance does not receive any requests in the specified period, it self-terminates. If no timeout period is specified, the UA instance times out after a pre-configured maximum wait period.

Session termination

To terminate a UA session, the client sends a termination request to the specific UA instance it wishes to terminate. After that instance has been terminated, the client should no longer communicate with the specified server and port. The port number may be recycled for future UA sessions.

Session control

To interact with the User Agent, a client makes HTTP POST requests to the assigned URL. The body of the request contains the method it wishes to invoke, along with any parameters required for that method. Available methods are listed below in Table 1.

5

Table 1

Method	Description
goToURL	**Parameters:** The absolute URL which the User Agent goes to. **Actions:** The User Agent goes to the URL specified. This method returns immediately. It does not wait until successfully fetching and rendering the page until it returns. This is to improve performance (the client does not get blocked an HTTP POST request while waiting for the User Agent to finish getting the page). **Returns:** Success or failure code
navigate	**Parameters:** XML encoding of a series of valid navigation-related actions. There are two types of actions that may be performed: *filling form fields* and *firing events* (see below for details). There are two "sub-methods" for navigate. • fillFormField *[field locations and values]* – Fills out the specified fields with the specified data. The client specifies the location of the field and the value to assign to that field. The location comprises the frame in which the target form resides and the element ID of the field in the frame. The client may specify multiple fields in multiple forms in multiple frames in a single request. • fireEvent *element_ID event [event parameters]* – Fires the specified event on the specified element with the specified parameters. Note: Clicking a link is also an event. **Actions:** The User Agent executes the actions specified. The client

	may specify zero or more of either type of action in the sequence of their choice. However, the UA only guarantees execution up to an event which causes a page transition. So, for example, if the client tells the UA to click the "Search" button on Google and, subsequently, tells it to enter some string in the text field, the UA will click the "Search" button, detect a page change and ignore the rest of the input. Thus, the client should specify actions in the order in which they are intended to be taken. **Returns:** Success or failure code
changedFrames	**Parameters:** None **Actions:** None **Returns:** Indicates if all frames changed or only some frames changed. Returns the IDs of the frames which changed since the last request in the latter case and all frame IDs in the former case.
getOrigHTML	**Parameters:** The ID of the frame of interest **Actions:** None **Returns:** The original, un-rendered HTML of the specified frame
getXHTML	**Parameters:** The ID of the frame of interest **Actions:** None **Returns:** The XHTML representation (with element ID's inserted in each tag) of the rendered HTML of the specified frame. Separate frames are requested in separate requests.
terminate	Terminates the UA instance

COM interface

The COM interface exposes two methods: *getXHTML* and *getIBrowser*. These are
5   described in Table 2.

Table 2

Method	Description
getIBrowser	**Parameters:** None **Actions:** None **Returns:** A handle to the COM browser interface IBrowser
getXHTML	**Parameters:** The ID of the frame of interest **Actions:** None **Returns:** The XHTML representation (with element ID's inserted in each tag) of the rendered HTML of the specified frame. Separate frames are requested in separate requests.

Design

Component overview

A complete UA server setup involves several components. Namely, the load balancer, the instance manager and the User Agent itself. Not all of these components are necessarily used at both design-time and runtime. The components are first enumerated and described individually. Subsequent sections describe how they are used at design-time and runtime.

Installer

The installer program should install all code necessary for the correct functioning of the User Agent. In addition, it should do any configuration necessary for the correct and efficient functioning of the User Agent. This includes, but is not necessarily limited to, the following:

- Changing the Internet Explorer settings so that images are not loaded by default.

- Setting the number of network handles and other server resources which may be required to optimize performance.

5

The Load Balancer

The load balancer is a standard web server hardware load balancer. It acts invisibly as a relay between clients and individual UA servers. It receives HTTP requests from

10 clients and routes those requests to the server with the least load and routes the responses back to the client. It determines load by the number of active HTTP connections each server has. Servers with more active connections are considered to have a heavier load.

15 This entire process is completely invisible to the client. The load balancer operates on OSI layer 3. Neither the client nor the server need have any knowledge of its existence. The domain portion of the URL to which the client makes requests will point to the load balancer. The load balancer then acts as an intermediary for that HTTP transaction, sending the response from the server back to the client. This is the only request for that

20 session which will be serviced by the load balancer. From this point on, clients will communicate directly with the User Agent instance assigned to them on the server which ended up servicing their rerouted request.

The Instance Manager

25

The instance manager is a daemon ("service" in Windows terminology) which spawns new UA instances when it receives an HTTP POST request to do so. It then informs the newly spawned UA instance and the client of the port which they should subsequently use to communicate with each other. It may choose any available port for this purpose.

CLIC1P019

It also passes the body of the HTTP request to the new UA instance. Any parameters which are to be passed to the UA are contained in this body.

This is the only request for that session which will be serviced by the instance manager. From this point on, clients will communicate with the UA instance directly via the assigned port. Once the instance terminates, the instance manager may recycle the port for use by another instance.

When it receives a request, the Instance Manager should check to see that sufficient server resources are free to service the request without severely degrading server performance (in extreme cases, causing thrashing or crashing the server). Memory is the most obvious resource in question. However, if any other resources turn out to be severe bottlenecks, they should also be checked. If sufficient resources are not free, the request should be denied and the client should be sent an informative message indicating why the request was denied.

The User Agent

The User Agent is the actual application that does the majority of the work. It does not run by default. The instance manager spawns new UA instances in response to requests. The high-level functionality and interface of the UA has been discussed above. To summarize briefly, the UA's main purpose is to fetch and render web pages and return their XHTML or XML representation to the client.

Design-time operation

Figure **4** illustrates a flow **400** of User Agent **402** operation at design-time. Design-time operation is relatively simple from a User Agent point of view. As shown, the IDT **404**simply instantiates the COM object UserAgent. The COM interface **406** exposes two methods (discussed earlier). getXHTML returns the XHTML representation of the

current page. getIBrowser returns a handle to the IBrowser object. The HTTP interface
408 is shown in dotted lines because it is unused at design-time.

In effect, the UA acts like a normal browser embedded in the IDT with the sole

5   exception that it offers a method to get the XHTML representation of the page (with
element IDs inserted in each tag). It is the responsibility of the IDT to use the browser
object to achieve its desired functionality. In particular, it can trap any and all events in
browser (including user events such as a right-clicking). It can also access the DOM
tree of the current page either for inspection or modification.

10

Runtime operation

Figure **5** shows the flow **500** of User Agent operation at runtime. Runtime operation
involves nearly all of the components used at design-time (although the COM interface

15   of the UA is not used). The load balancer receives client requests and routes them to
quickest server **502**. The instance manager **504** runs as a daemon on the UA servers.
Upon receiving a request, they spawn a new UA instance. Clients **506** then
communicate directly with the this new UA instance. The COM interface is shown in
dotted lines because it is not used at runtime.

20

Pattern Recording and Replay

The Platform gives its users the ability to create (record) functional elements called

25   patterns and use those elements (replay) in their custom applications. A pattern consists
of a collection of states (see *Glossary*) and states can be of different types. For
example, an action that requires accessing the World Wide Web can be represented by
an XML (or other type) state and an action that would require sending e-mail can be a
part of an SMTP state. Each type of state has a dedicated connector, which is used to

create the state definitions and operate on those states. Thus, an XML Connector Module (XCM) is the connector that aids in recording and replaying of XML states. A database connector is used to store and retrieve data from a database, but may not point to a next state. An XML Connector points to a next table. For example, the XML

5   Connector in state A would instruct the UA to execute action X to access state C. Note that the PRE deals with any state, such as a database state (e.g., Yahoo inbox stored in a database).

Figure **6** is a flow chart **600** that illustrates a method for recording a pattern using the

10   User Agent. In particular, the user performs the desired navigation and associated input, which are recorded. In operation **602**, the user is sent a request for a URL. Upon receiving the URL from the user, the URL is sent to the User Agent in operation **604**. The User Agent parses the request in operation **606** to determine a meaning of the request. In operation **608**, the User Agent retrieves the web page associated with the

15   URL and sends it back to the user. This (and other) web pages are represented in the pattern as States. In operation **610**, actions are received from the user. Such actions can include selection of links, input of data in fields, etc. The User Agent receives the user input in operation **612**, and in operation **614**, executes the actions on the remote website. The actions are recorded in operation **616**. The resulting web page is shown

20   the to the user in operation **618**. The user then identifies the parts of the page he or she is interested in, which is received in operation **620**. The identified parts are recorded in operation **622**.

The system now knows how to retrieve desired information using the pattern.

25   Accordingly, all users using the system can obtain that content. The next (or same) user that wants to perform a similar action enters the URL and the pattern is replayed with the User Agent executing the actions received of operation **614**. The parts of the web page that were identified by the user and stored in operation **620** are returned to the user. The user can also define a default area to access in the event that a state is not

CLIC1P019

recognized. See also the description below regarding state identification based on content and probability.

The information stored in a pattern may involve (i) the web or non-web interfaces
5 presented to the user, (ii) the actions performed by the user or the system in the submission of the information, (iii) the parameters required to complete the transaction and submitted by the user, (iv) the automatic or manual navigation of the user within the transaction process, and/or (v) the content or results returned by the system or selected by the user.

10

A transaction pattern for any transaction need only be recorded once by anyone with sufficient system access privilege. Once recorded, any user may conduct that transaction, any number of times, by the invocation of the recorded transaction, or "transaction macro."

15

Since a transaction macro can be stored in any fashion and invoked, directly or indirectly, by any agent (human or automated), it enables the completion of the transaction it describes in any fashion from any device.

20 As an example of use, suppose the user goes to Amazon.com and wants to search for books about Java. A User Agent performs a search for the books and a predetermined number of results are output. The Buy and Buy Later buttons are output along with the results. The user is allowed to select one of the books. The User Agent retrieves additional information about the book (by following a link, for example). If the user
25 clicks on the Buy button, the User Agent follows a pattern to purchase the book automatically by filling in the appropriate information, including pre-stored user address information and credit card information. A Purchase Completed page is shown to the user stating that the book has been purchased.

Exemplary Pattern Recording Interface

According to a preferred embodiment of the present invention, pattern recording is done through an intuitive interface that allows the user to quickly and easily record patterns.

5   More particularly, the interface allows creation of an application for retrieving a variable from a database as well as creation of an interface for displaying the variable on a particular type and/or model of device.

Figure 7 illustrates a recording interface **700**. As shown, a grid screen **702** is provided.

10   It is here that the graphical representation of an application for retrieving data from a data site is displayed. An Object Bar **704** allows insertion of connectors and shapes into the grid screen. A Catalog Bar **706** displays several types of specialized connectors and components. One specialized connector is the database connector **708** representing a connection to a database. Selection of the tabs on the Catalog Bar displays different

15   connectors and components. The items in these bars can be dragged and dropped into the grid screen and connected with connectors from the Object Bar. The Project Bar **710** shows the names and types of variables being manipulated. Variables can be added by using a wizard, which starts upon selection of the wizard button **712**.

20   The following example illustrates creation of an application for retrieving a customer identifier (ID) from a database and creating an interface for displaying the customer ID on a particular type and/or model of device. As will be discussed in the example, the recording interface allows creation of different interfaces for different devices based on the capabilities of each.

25

Referring again to Figure 7, a start screen **714** is displayed upon selection of the wizard button. The database server is selected from the drop down menu **716** and the Next button is selected. A database selection screen (not shown) is presented, which allows the user to select the desired database from which the desired data is to be retrieved.

Figure **8** illustrates a mode selection screen **800**. Here, the user is allowed to select step by step mode or advanced mode. The desired mode is chosen and the Next button is selected. In this example, the step by step mode is shown.

5

Figure **9** shows a field selection screen **900**. The tables of the selected database are presented in the drop down menu **902**. The user is allowed to select which table to view. The fields of the selected table are presented in the fields display portion **904**. The user is allowed to select one or more fields from the field display portion. Note, multiple fields can be chosen using CTRL + R-click. The Next button is selected.

10

Figure **10** depicts a filter screen **1000**, which allows entry of filtering elements used to restrict the variables retrieved from the database. As shown, the field chosen in the field selection screen is displayed in the Field column **1002**. Operands can be entered into the Operator column **1004** (or chosen from a drop down menu). Illustrative operands include >, <, ≥, ≤, and =. The Value column **1006** receives comparison values, to which the variables associated with the field are compared using the operand. The Next button is selected.

15

Figure **11** illustrates a results screen **1100** displaying the results of the query in a results portion **1102**. The query is also displayed in a query portion **1104**. The query may be edited in the query portion to change the results returned. The Execute button **1106** is selected to re-execute the query and retrieve results of the edited query. The Next button is selected.

20

25

Figure **12** illustrates an assign variable screen **1200**. Here, the results are assigned to a variable and saved for later use. The name of the variable is entered in the Variable Name field **1202**. In this example, the variable is named "customer." The Next button is selected.

Figure **13** shows the grid screen **700**. The new variable "customer" now appears in the
Project Bar **710**. A Database Connector **1302** is dragged from the Catalog Bar, as is a
Select Query object **1304**. A connector **1306** is dragged and dropped from the Object

5   Bar to the grid screen and used to connect the Database Connector and the Select Query
object.

Figure **14** shows the grid screen with additional objects added. Here, a Start Process
object **1402**, an Output object **1404**, an End Process object **1406**, and connectors have

10   been added.

Figure **15** illustrates a display specification screen **1500** from which the user can specify
the manner in which variables are displayed on particular devices. As shown, various
devices are shown in a tree structure. A type of device is selected, here the Palm V

15   version of the PDA. A representation **1502** of the PDA is displayed. The identifier
"customer" **1504** is added to the simulated display **1506** of the PDA. The variable field
1508 is also added to the display. During runtime, the actual result of the query is
displayed in place of the identifier and variable field. Text may also be added for
output in this screen. Here, the words "Customer ID:" **1510** have been added.

20

For example, a website may have a table having an output with five columns. On a
PDA, perhaps only four columns can be displayed on the display. The output can be set
to only show four variables. Similarly, if only two columns of the table can be
displayed on a phone, the output van be set to show only two variables.

25

Pattern Replay Engine

The Pattern Replay Engine (PRE) is the component of the platform that is responsible
for replaying a pattern from one particular state to another during runtime.

Figure **16** gives a general overview of how the PRE **1602** interfaces with other major

components including the Input Handler **1604**, session manager **1606**, connector **b0008**

(discussed in detail below), and TRE **1610**.

5

The PRE is a request-driven component. The PRE takes the request to make a

transition from a particular State. Transitions from one State to another are made by

executing Actions. The PRE communicates with the appropriate Connector (of the

current State) to execute the current Action in the request and get the new resulting

10   State. Next, all Script associated with the State, except for OnDestroy(), is evaluated.

The PRE then calls the Transcoding Page Rendering Engine (TRE) to "flush" its output

to the device. This means that any Content, which has been queued for display in the

TRE (while processing the request), is sent to the device. Every state can potentially

15   generate output to be displayed on the device, including internal states. This is a

desirable feature as it means the designer can send a status message to the device, e.g.

"Please wait while processing.." etc. The TRE may have to be reset every time, i.e.

after a "flush" which flushes any queued output to the device, the TRE has to be ready

to start building up a whole new display.

20

Finally, the State's OnDestroy() script is called, which can initiate further State

transitions from this script method if necessary.

Figure **17** is a flowchart of a process **1700** for replaying a pattern for transitioning from

25   one state to another state, according to an embodiment of the present invention. A

request to transition from a current state in a pattern to a new state is received in

operation **1702**. A state can be defined as the state of a remote application as defined by

the user. A pattern is a collection of states to be used as one functional element in an

application. In operation **1704**, information about transitioning between the states from

the current state to the new state is retrieved from the pattern. Scripting code of the current state is evaluated in operation **1706** to determine what action or actions are to be performed/executed in order to effect the transition to the new state. In operation **1708**, an action (or actions) is executed using the scripting code for effecting the transition

5   from the current state to the new state. An example of an action is clicking on a link. The new state is rendered and output to a user device in operation **1710** and **1712**, respectively.

Preferably, the action is executed by communicating with a connector in

10   communication with the appropriate state for executing the action. Also preferably, each type of state has a dedicated connector.

In another aspect of the present invention, each of the states generates output for display on the user device.

15

In a further aspect of the present invention, a further state transition is initiated and the process is repeated. Preferably, a history of states in the pattern that been traversed is maintained. Ideally, a state traversed out of sequence is detected. If it is determined that the incoming request is "out of Step", then a determination is made as to what point

20   in the history the request is coming from. Then a special Back event is sent to the appropriate Connectors in order to "back up" to that State. For example, Step 1 could represent the login page for Yahoo mail. The user then proceeds to the Inbox page (Step 2), and subsequently to a specific mail message (Step 3). If the user clicks "Back" twice on their device browser, the browser could simply render the cached pages for

25   Step 2 and Step 1. Then the user could initiate another login from Step 1. The present invention would detect that the user has gone back in sequence (since the Step ID is part of the request) and would respond as described above.

CLIC1P019

In one embodiment of the present invention, the transaction pattern may include a record of: information submitted by a user, actions taken by the user, actions taken by a system to generate results, and results sent to the user. As an option, the transaction pattern may include a record of actions taken by the system which enable access of the user to data, and actions enabled by the data to retrieve content. Storage may take any form with sufficient identification and description of any required step in the transaction process.

Figure **18** depicts a process **1800** for replaying a pattern for transitioning from one state to another state in a remote application, according to another embodiment of the present invention. In operation **1802**, a request to transition from a current state in a pattern to a new state is received. The states correspond to current and new states of a remote application. In operation **1804**, an action is executed for effecting the transition from the current state in the remote application to the new state in the remote application utilizing the pattern. An example of an action is clicking on a link. Note that the action can be executed by a user or by script. An output from the new state is generated and output to a user device in operations **1806** and **1808**.

In one aspect of the present invention, the action is executed by communicating with a connector associated with the state for executing the action. Preferably, each type of state has a dedicated connector. Also preferably, each of the states generates output for display on the user device.

A further state transition can be initiated. Likewise, a history of states in the pattern that been traversed can be maintained. Preferably, a state traversed out of sequence is detected.

During use of the present invention, such transaction pattern may be retrieved from memory when desired. Thereafter, the transaction pattern is executed to carry out the

transaction in an automated manner. An execution, or playback may include the invocation of a stored transaction pattern.

The various operations that may occur during execution of a transaction pattern may involve: (i) a direct or indirect retrieval of the transaction pattern by an automated and/or programmable agent, i.e. a computer server/client, an active or dormant program, a human being, etc. (ii) an automatic or user prompted submission of the required parameters in every step of the transaction process, (iii) the automatic navigation within the transaction process, (iv) a retrieval of the content returned by the system or specified by the user, and/or (v) the relaying of the content back to the user.

Figure **19** is a flowchart of a method **1900** for identifying a dynamic element and replaying a dynamic event according to an embodiment of the present invention: This method is preferably carried out by the User Agent in conjunction with the Pattern Replay Engine and State Recognition Module. A web browser is hosted in operation **1902**. In operation **1904**, a web page of a network site is rendered using the web browser so that it is in the form the user would see it. This allows the User Agent to identify dynamic elements based on its visual properties, including text labels, positioning, etc. A dynamic element with pre-specified properties is identified on the rendered web page in operation **1906**. Such an element could be a "Buy" button on the web page. In operation **1908**, the identified element is executed based on a prerecorded pattern. A web page resulting from execution of the element is output in operation **1910**. This method overcomes the disadvantages of attempting to understand the source (e.g., HTML) of the content. Rather, particular elements can be identified by their properties, even though the elements have changed form. For example, the "Buy" button may have changed to a different type of icon or even an animated graphic. Regardless, the present invention is able to identify the element and execute it.

More information regarding pattern recording and replay is provided in co-pending
Provisional U.S. Patent Application entitled SYSTEM, METHOD AND COMPUTER
PROGRAM PRODUCT FOR THE RECORDING AND PLAYBACK OF
TRANSACTION MACROS, filed April 12, 2001 under serial number 60/283,781 and
5   assigned to common assignee, Clickmarks, Inc., and which is incorporated herein by
reference

Connector Plugin Support

10   Figure **20** illustrates a Connector class hierarchy **2000**. The Platform uses Connectors
to provide specific functionality for interacting with a remote application. See *State*
Recognition Module and Connector below. A standard interface **2002** is defined for the
Connector and its associated classes **2004, 2006** such as Connector-specific State
information classes.
15

Figure **21** depicts a State class hierarchy **2100**. In the XML description of the Pattern,
all possible States are defined, including State information which is generic and State
information which is specific to the Connector.

20   An interface **2100** is defined. A public StateImpl (implementation object) **2102** is
provided to subclass from. All Connectors have a "createState()" method which creates
a State object which is *specific* to the Connector. This makes the design clean so that
the PRE does not need to instantiate Connector-specific State objects (e.g.
HTMLStateImpl **2104**) itself.
25

Scripting

The PRE evaluates scripting code associated with each State. The user's script can be written in JavaScript (which is specified by the designer using the IDT). The script has access to the Platform's Scripting API.

5   Script Event Methods

There are five preferred Scripting methods which are called when a State is entered:

OnInit(...): method called when the PRE first enters the State. Used to perform
10   any initial processing.

OnData<contentID>(): group of methods called to extract and manipulate the content Data retrieved by the Connector as it becomes available. There is a separate 'OnData' function for each content data item of interest. Each
15   JavaScript function name is distinguished by a suffix consisting of the content ID. Using the Scripting API, the Data can be stored in Variables and/or in Content UI objects for rendering on the device.

OnDataComplete(): single method called when all the data content retrieval has
20   been completed.

OnDestroy(): last script method called. Typically used for cleanup and to explicitly execute one or more Actions and/or to initiate a State transition (using the Scripting API). See the *Executing Actions Via Script* section for more
25   details.

OnError(): method called when PRE is unable to normally process the State for any particular reason. (It does not get called when an Action-execution results in a new State – in this case, the 'catch' code within the script gets called.)

Invoking the Transcoding Page Rendering Engine (TRE)

The TRE's 'flush' method is called to output to the device any content that has been
5   rendered. This method is called every time a State transition occurs. Specifically, it is
called after the OnDataComplete() method but before the OnDestroy() method.

Executing Actions Via Script

10   In the OnDestroy method, the user can explicitly execute an Action.

Exposed Scripting Functionality

In addition to having the core JavaScript language available, the Scripting code can take
15   advantage of powerful functionality via the Scripting API and by importing and calling
methods on special Scripting objects provided by the PRE.

Scripting API

20   <u>Variable Access</u>
        •   variable retrieval – reading in User, Session, and Application variables
        •   variable storage – writing out User, Session, and Application variables

        <u>Pattern Flow Control Methods</u>
25   •   Action execution – executing any Action within the current State
        •   State transition – jumping to any State within   the current Pattern

Importable Java Objects

Data Retrieval: (class: DataRetrieval)

- provides access to the Data of interest associated with the current State.

Content Object

5
- allows script to put data into a Content Object, which in turn has a render method which can be called to output the Content Object to the device.

Variable Management

10 The PRE handles requests to read and write out Variables. These requests occur from within the scripting code (See *Scripting API*). The requests can be for any of the three different types of Variables: User, Session, and Application. Variable read/write requests are initiated from the scripting code. The scripting code gains access to these variables through the ScriptingAPI's "Variable Access" methods. (See *Scripting API*

15 section.)

Pattern Replay Engine Tasks

Accepting the Request

20

The PRE accepts requests from the Input Handler (e.g. HTTP Input Handler). The PRE receives the Pattern ID, Step Number, Event ID, and the data parameters to successfully process the request. The data parameters are passed in a Java Hashtable object containing name-value pairs.

25

Determining Corresponding State from Step Number

The PRE maintains a history of the States in the Pattern that have been traversed as a monotonically increasing sequence of Steps. The PRE maintains a mapping between each Step and its corresponding State.

5 *Detecting a Step Number out of Sequence*

Since the request given to the PRE contains the Step Number of the originating State, the PRE determines whether the user is traversing the States in a linear fashion, or if the user has gone "back" in the history of Steps, by comparing the request Step Number

10 with the Step Number of the last-traversed State. If the user has gone "Back" in the sequence, this is a special case. See the *Going Back in Sequence* section.

Retrieving the Session

15 The PRE communicates with the SessionManager to obtain the Session object for a request. This Session object contains a PatternContext in particular, which is discussed below.

Maintaining the PatternContext

20

The PatternContext object is contained within the Session object. It represents the runtime information associated with a particular Pattern and Session, including the current Step number, the mapping between Steps and States, the collection of Connector objects, and the Session variables. The PRE keeps the PatternContext up-to-date and

25 stores it persistently (via the SessionManager) by storing the Session object that contains it.

Retrieving the Pattern, State, and Script

The Pattern object contains all of its State objects. The PRE requests the specified State from Pattern object by passing it a State ID.

The State object contains a Script object, which represents the scripting code that is
5   associated with the State. (There is only one Script object per State object.) The PRE requests the Script object from the State object.

Instantiating the Connector

10   Each State can only be handled by one particular type of Connector. Each Connector is instantiated only as needed since a Pattern may never branch to a particular Connector.

The PRE asks the State object what type of Connector to use. The PRE first tries to retrieve the Connector from the PatternContext object if it already exists. If the
15   Connector has not yet been instantiated, the PRE calls a createConnector method, passing it the type value and Pattern reference to associate with the Connector. The createConnector function creates the appropriate Connector subclass instance which corresponds to the specified type and returns it to the caller (the PRE).

20   *Extracting Connector-specific Data*

The XML definition of each State within a Pattern contains information that is Connector-specific. The PRE does not need to be able to interpret this Connector-specific information; it simply needs to pass this information (for each State) to the
25   Connector when an Action is to be performed. (See *Invoking the Connector.*)

Invoking the Connector

The PRE invokes the Connector (determined above) by calling its executeAction(...) method. This method takes the Action and a Hashtable of data parameters as name-value pairs. The Connector has access to the Pattern using a reference passed during the Connector's construction. The Connector only has access to the Connector-specific

5   Pattern information pertinent to this particular Connector subclass instance. For example, the XML Connector stores State-property information within the Pattern which is specific to the XML Connector only. The Connector returns an ActionResult object which has methods to get the resulting State ID and retrieve all of the content data items of interest. If any errors occur (such as not being able to process the Back

10   event), an exception is thrown within the executeAction method.

Predetermined Transitions Between States

At design time, the user can specify a State transition that should be performed after a

15   State's script code has finished evaluating, called a predetermined State transition. The IDT specifies this in the Pattern XML. The PRE recognizes such a predetermined State transition in the XML and performs the transition after all other script code (not including OnError) has been evaluated for this particular State. And unlike State transitions which are specified in the script code, predetermined State transitions are

20   preferably implemented directly from the PRE's own Java code, not JavaScript.

Evaluating Script

Assuming no error occurred while performing the transition, the PRE evaluates script

25   associated with the new resulting State. See the separate *Scripting* section for details.

Going Back in Sequence

If the PRE has detected that the incoming request is "out of Step", then it finds at what point in the history the request is coming from. Then it sends a special Back event to the appropriate Connectors in order to "back up" to that State. For example, Step 1 could represent the login page for Yahoo mail. The user then proceeds to the Inbox

5 page (Step 2), and subsequently to a specific mail message (Step 3). If the user clicks "Back" twice on their device browser, the browser could simply render the cached pages for Step 2 and Step 1. Then the user could initiate another login from Step 1. The PRE would detect that the user has gone back in sequence (since the Step ID is part of the request) and would respond as described above.

10

Scripting

The user can specify script to be evaluated when a State is encountered. (This script is specified using the IDT.) Using script (which is specifically JavaScript), the user has

15 access to the Platform's Scripting API and Scripting Java objects. The *Functionality* section describes the functionality that is exposed.

Scripting Implementation

20 Figure **22** is a diagram **2200** illustrating how different components interact to support Scripting. PRE preferably supports JavaScript scripting using an open-source platform **2202** from Mozilla called Rhino. Rhino is Mozilla's Java implementation of the core JavaScript language. Using Rhino, the PRE **2204** can evaluate JavaScript from within Java objects. In addition to the core language, Rhino has implemented JavaAdapters,

25 which allow JavaScript to implement any Java interface or extend any Java class with a JavaScript object.

The Scripting Engine **2206** receives a request to evaluate a particular script inside a Script object. The Scripting Engine receives a reference to the content DOM tree so that the JavaScript can manipulate it. The Scripting Engine communicates with the Session Manager **2208** to fulfill any Variable read/write requests that it made available

5 via the Scripting API.

State Recognition Module and Connector

10 The XML Connector is a 'state-keeping' connector. That is, it is always in some specific state. That state corresponds to some state of the remote application (see *Glossary*). A transition from one XML state to another cannot usually be made, unless it is done by using an "action" that will change the state of the remote application and thus change the state of the connector accordingly.

15

The XML Connector identifies a remote state by the output provided by the remote application in that state. In fact, the remote application may have much more information about its current state then what it displays to its users. Thus, a distinction can be made between a local state ("state") – which will refer to the state assigned by

20 the user to the remote application state – and the actual remote state. The transition from one remote state to another is a function of the user input and the state of the remote application. Since the XML Connector does not have all information about the remote state, it cannot accurately know what the next state of the remote application will be, until the remote application provides output corresponding to that state. Moreover, if at

25 one time certain user action in state A caused the remote application to transition to state B, it does not follow that the same transition will occur the next time the same user action is executed in state A. In light of the above discussion, the replaying mechanism preferably recognizes the state by its current output, not by the previous state and transition. This is done with the help of the State Recognition Module (SRM).

The SRM creates a state's "output structure" description for a state and then uses that description during replay to recognize a state. The term "output structure" refers to the window and frame structure of the output from the remote application. The output

5   structure description contains the information about how many windows were presented to the user at a given state in remote application and how many frames were in each window and the nesting structure of those frames. (SRM uses the CRM to recognize individual documents in a document structure.)

10   **Functionality**

Figure **23** depicts a process **2300** for recognizing a state based on a current output, according to one embodiment of the present invention. In operation **2302**, current output is retrieved from a remote application utilizing a network, the current output

15   corresponding to a state of the remote application. An output structure description of the remote state is generated in operation **2304**. The output structure description is preferably of a window and frame structure of the output. The output structure definition is compare to a pre-defined state definition of a recorded state (called a state description) in operation **2306**. In operation **2308**, a determination is made as to

20   whether the remote state corresponds to the recorded state. Note that the recorded state definition is created during recording of the pattern.

As an option, predefined information (e.g. a StateData Object) about the remote state can be returned if the remote state corresponds to the recorded state. The predefined

25   information can then be used to extract data from the remote application.

In one aspect of the present invention, the output structure definition includes a number of windows presented in the remote state, a number of frames in each window, a nesting structure of the windows, and/or a nesting structure of the frames. Preferably, the

recorded state is part of a pattern pre-recorded by a user. In yet another aspect of the present invention, content of the remote output is recognized based on recorded content properties and used to help recognize the state. Preferably, the recognized content is utilized for recognizing the state based at least in part on a weighted average of content

5   recognition and content importance. Note that the content can be one or more documents, one or more content items in a document(s), etc.

Figure **24** is a flow diagram of a process **2400** for recognizing a remote state utilizing individual document recognition, according to another embodiment of the present

10   invention. This embodiment identifies the state of a site or page by the content of its content (such as documents) instead of (or in addition to) the layout structure of these documents. In operation **2402**, a plurality of documents are received. An attempt to recognize each of the documents as being associated with a predetermined state is made in operation **2404**. In operation **2406**, a state is identified based on the recognition of

15   the documents.

A Content Recognition Module (CRM) can be used to recognize states. Basically the state consists of a bunch of documents, each one of which is first recognized by the CRM. If the state consists of documents A, B and C, then if the CRM recognizes all of

20   A,B and C from the documents given it by the SRM, the state is recognized. Otherwise, if the CRM recognizes a different set of documents, then the state is recognized as the state containing that set of documents, or no state is recognized if no state corresponds to that set of documents, the pattern is placed in an error state.

25   Preferably, if more than one pattern state matches the remote state, a comparison of output structures of the remote state and pattern states is performed, as set forth above.

In one aspect of the present invention, at least a portion of the state definitions are given a weighting based on a probability that they will occur when in another state. Preferably, the

weighting is based on a state resulting from execution of a prior action. For example, if during last replay (or record) of a pattern, action *x* executed in state *A* resulted in state *B*, then during current replay when *x* is executed in *A*, *B* will be the most probable resulting state, more so than any other state in the pattern. The weight is used to optimize the state

5   recognition algorithm by comparing the state to be recognized first with the state most likely to occur after executing the action in the previous state. If that state does not match the state to be recognized, then the next most likely state is compared, and so on. Content item weights are defined on the document as it is part of the state, but they are not defined on the state itself.

10

Figure **25** is a flow diagram **2500** that illustrates the interaction between the XML Connector Module (XCM) **2502** and the Pattern Replay Engine (PRE) **2504**. As discussed above, the PRE performs the task of replaying (executing) a pattern at run-time. It goes through each state of the pattern, and depending on the type of the state

15   invokes the appropriate connector. The XML Connector Module (XCM) is invoked to execute XML states. Its input is the request parameters (DataParams) and definitions for all possible XML states in the current pattern, and it outputs a StateData object, which can be used to obtain information about the next state and the desired data elements.

20   Figure **26** is a flow diagram **2600** of the operation of the XML Connector **2602**. The request parameters contain the action to be executed and other element changes caused by the end user (such as filling out a form field). This information is sent to the User Agent Module **2604**, so that it fetches the output of the resulting remote state. That output is then sent to the SRM **2606** so that the current remote output can be matched to

25   a state definition created during design time. The SRM accepts the output of the remote state and the properties of all possible local states to which this remote state can correspond. If the remote output is recognized as one of the recorded states, a StateData object is returned which contains the information about the recognized state and can be used to extract the desired content and other information stored during recording.

At design-time, the XML connector is not used, because the task of fetching the output of the remote application is performed directly by the Interactive Design Tool (IDT), which is the user interface for the pattern recording. However, the IDT may still use the

5   SRM to create state definitions for pattern states. These state definitions contain the information that is used later during replay to identify remote states and match them to local states (which are defined by these state definitions).

Figure **27** is a flow diagram **2700** of SRM and IDT interaction (requests executed from

10   top to bottom). During design time, the IDT **2702** sends requests to the SRM **2704** to calculate state properties for some remote output. That request contains the remote output XML, the element id's for XML elements of interest in that output and the document and content descriptions created by the user to aid in identifying documents and content elements. After analyzing the state structure, the documents and the content

15   elements, SRM returns to the IDT the properties, which are used during run-time to identify the state. If a state with the same structure and document properties already exists, this information is returned to IDT to fetch more information from the user on differentiating the state from the other similar state (if that is desired).

20   Besides calculating the state properties, the SRM is also able to identify if the new XHTML output matches any of the existing state properties for a previously defined state. This allows identification of any identical states in the pattern, which may be difficult to differentiate at run-time. It will also aid the designer if a state is to be edited after being recorded initially. That is, if the designer points the IDT browser to a remote

25   state that was already identified, the system will recognize that this is an already existing state and will let the designer modify content and actions at that state.

In one embodiment of the present invention, the interaction between the IDT and the SRM proceeds as follows. Once the designer loads the XHTML output for a given

remote state into the IDT browser, that XHTML is sent to the SRM to compare against the properties of the states that already exists in the pattern (unless no states exists). If the XHTML matches one of the state properties, the state id for that state is returned.

5   Once the user has identified all the content of interest in a state and has executed an action that causes a state transition (DOM modifying action or page transition action), the element ids are sent to the SRM to create content properties.

The following is a sample interaction between the SRM and IDT:

10

    IDT:

    Add A State (check existence):

15   <SRM:STATE id="2">
        <SRM:DOC id="21" type="window" rank="necessary">
            <SRM:DOC id="211" type="frame" rank="necessary">
                <SRM:DOC id="2111" type="frame" rank="necessary">
                    <HTML>...</HTML>
20   </SRM:DOC>
                <SRM:DOC id="2112" type="frame" rank="necessary">
                    <SRM:DOC id="21121" type="iframe"
            rank="ignore">
                        <HTML>...</HTML>
25   </SRM:DOC>
                  <HTML>...</HTML>
                </SRM:DOC>
            <HTML>...</HTML>
        </SRM:DOC>

```
                    <HTML>...</HTML>
                </SRM:DOC>
            </SRM:STATE>
```

5   **SRM**:


```
        <SRM:RESP status="ok"></SRM:RESP>
```


IDT:

10

Add content to state.

```
<SRM:STATE id="2" status="new">
    <DOCUMENT id="21" url="http://www.yahoo.com">
        <CONTENT id="23" rank="necessary">
15          <PROPERTY class="ATTR_DEF" rank="necessary">
                <PARAM name="href" value="http://mail.yahoo.com">
                </PARAM>
            </PROPERTY>
        </CONTENT>
20  </DOCUMENT>
    <EVENTSTREAM>
    <EVENT id="1">
        <ACTION>ONCLICK</ACTION>
        <CONTENTID>23</CONTENTID>
25      <DOCUMENTID>21</DOCUMENTID>
        <EVENTHANDLER>on_submit_page</EVENTHANDLER>
    </EVENT>
    </EVENTSTREAM>
</SRM:STATE>
30
```


SRM:

35   `<SRM:RESP status="ok"></SRM:RESP>`

If the SRM informs the IDT that the state already exists, the IDT may ask some document description details from the user (for example, the user may specify that this document should be identified by a certain text which is not present in the documents in the other state which is similar to this one). The SRM will try to reanalyze the state output given the additional user input and if the additional information helps to differentiate the states, the state will be added as a new state.

XML Applications

Most common XML applications are HTML web-based applications and this connector supports these applications. There are two distinct issues that the connector has to deal with in supporting XML applications. These are data representation (state recognition and content retrieval) and transition actions. State recognition and content retrieval are supported on any XML document or set of documents (multiple documents are possible in one state only through the use of HTML windows and frames). The transition actions supported are an arbitrary user event (as represented by the event object) and GET and POST requests to an arbitrary URL.

The XML Connector interoperates between the Pattern Replay Engine, the User Agent and the State Recognition Module.

Figure **28** depicts a program flow **2800** of the Connector. As shown, a request to execute an event or go to an initial state is received from the PRE in operation **2802**. In operation **2804**, the event is sent to the User Agent. A response is received from the User Agent in operation **2806**. If an error is found, the error is returned to the PRE. If no error is found, the current output and state definitions are sent to the SRM in operation **2808**. In operation **2810**, a new state id is received from the SRM. If an error

is found in the state id, it is send to the PRE. If no error is found, the new state id and content retrieval object are sent to the PRE in operation **2812**.

Additional Embodiment of the SRM

5

While sophisticated techniques can be used to recognize a remote state by its output structure, an embodiment of the present invention identifies a state by the content of its documents rather than by the layout structure of these documents. So if on day a site navigation menu is in the left frame and the contents of the site is in the right frame and

10 the next day it is the opposite, that does not have any impact on the functionality of the application at that state. In light of the above, while the SRM has the information about the exact structure of the state's output, state recognition occurs based only upon whether the required documents are present in the unordered list of the documents output at the current state. So, state recognition most heavily relies upon individual

15 document recognition (performed by the CRM).

The SRM works according to this algorithm:

- Send all documents from the current output to the CRM for document
20 recognition
- Compare the list of document ids in the definition of each state in the pattern (of the proper type) with the list of document ids in the current output.
- If a match is found, state is identified. (See below on how matching is done.)
- If more then one match is found, then output structures are compared for the
25 matched states.

There are two additional features in the SRM, which provide some more intelligence for state recognition. One is the Transition Probability List (TPL). This list provides an ordered array of state ids (which have higher probability than others to match the

current output) where the first state id corresponds to the state that will be the most probable to match the current output, the second state id corresponds to state which is the second most probable state to match the current output and so on. The Pattern Replay Engine can create this vector by keeping a history of which states resulted from

5 a certain action being executed in a certain state. So if during last replay (or record) action x executed in state A resulted in state B, then during current replay when x is executed in A, B will be the most probable resulting state, more so than any other state in the pattern. During each replay the resulting state id can be saved for any state-action combination and that information can be used later to create the transition probability

10 list.

The transition probability list is used by the SRM to generate hints for document recognition, which are sent to the CRM. The hints are provided in an ordered list of document ids where the position of document ids implies their probability. Thus, the

15 first document id is the most probable to match the current document, the next one has second highest probability and so on (similar to the transition probability list). The output structure is used to generate hints. Each state id in the transition probability list points to state properties. The structure of the each state's output, as given by the properties, is compared with the current output structure being analyzed, and if a

20 document's tree position matches the position of a document element in state properties, the id of that document element is sent as the most probable to match that document. The next most probable ids are the ids of documents on the same level of the tree and the next are all document ids given by the state property. If tree positions do not match, then the list of all document ids in the more probable states will be sent as more

25 probable to match each document in the current output.

Example:

If pattern contains two state properties as follows:

CLIC1P019

```
<SRM:STATE id="1">
        <SRM:DOC id="11" type="window" rank="necessary">
            <SRM:DOC id="111" type="frame" rank="necessary">
5                   <SRM:DOC id="1111" type="frame"
rank="necessary">
                    </SRM:DOC>
                    <SRM:DOC id="1112" type="frame"
rank="necessary">
10                  </SRM:DOC>
                    <SRM:DOC id="1113" type="frame"
rank="ignore">
                    </SRM:DOC>
            </SRM:DOC>
15      </SRM:DOC>
    </SRM:STATE>

    <SRM:STATE id="2">
        <SRM:DOC id="21" type="window" rank="necessary">
20          <SRM:DOC id="211" type="frame" rank="necessary">
                <SRM:DOC id="2111" type="frame" rank="necessary">
                </SRM:DOC>
                <SRM:DOC id="2112" type="frame" rank="necessary">
                    <SRM:DOC id="21121" type="iframe"
25              rank="ignore">
                    </SRM:DOC>
                </SRM:DOC>
            </SRM:DOC>
        </SRM:DOC>
```

</SRM:STATE>

The transition probability list is: TPL = "2,1".

5   If current step output received has a window with two frames inside, the hints list for the first frame will be an ordered list "2111, 2112, 211, 21, 21121, 1111, 1112, 1113, 111, 11".

10   Another feature is the *document importance ranking*. This enables the designer to provide a rank to each document signifying how important its presence is to identifying the state. The ranking is specified on the scale from 1 through 5, where 1 means that the document must be present for the state to be recognized and 5 means that the presence of the document has no bearing on state recognition (2, 3 and 4 are for additional flexibility and can be used for more sophisticated state recognition). The IDT provides

15   the designer an ability to provide importance ranking for each document in a state as part of document description. If such ranking is not provided, all documents with any identified content elements are given a ranking of 1, otherwise a document is ranked with a 5.

20   While various embodiments have been described above, it should be understood that they have been presented by way of example only, and not limitation. Thus, for example, the Connector described above should not be understood to only execute XML, but should be understood to also execute Hypertext Markup Language (HTML) and Standard Generalized Markup Language (SGML).

25

Therefore, the breadth and scope of a preferred embodiment should not be limited by any of the above-described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.

CLIC1P019